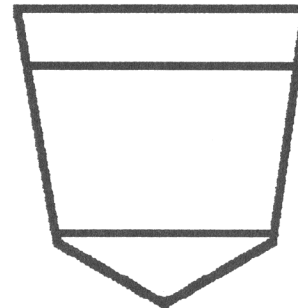


POCKET COMPUTER NEWSLETTER



© Copyright 1985 POCKET COMPUTER NEWSLETTER

Issue 43 - November

— FOR PC-1250/51/60/61 USERS —

AN ADDENDUM TO THE PC-1250 MANUAL

Rick Wenger, 6221 - 18th Avenue, Kenosha, WI 53140, thinks the "Student Computer Handbook" packed with the PC-1250 could use the following updating.

Strings and Stuff

Subscripts for one or two dimensional arrays may go from the value 0 to 255, not just 1 to 255. Thus, a statement such as `DIM C(4,3)` sets up an array of 5*4 elements. It consequently uses up a total of 5*4*8 bytes plus 6 bytes for the array header information, yielding a total of 166 bytes. This is far more than the 96 bytes indicated on page 96 of the manual.

The "extended memory" A-subscripted variable can be dimensioned in the implied manner of: `A(29)=expression`. In fact, just asking the computer to `PRINT A(29)` has the effect of creating three new variables: `A(27)`, `A(28)` and `A(29)`. Implied dimensioning of this type also has the effect of creating six header bytes. Thus, the number of bytes used in this example would be 3*8+6=30. At this point (unlike a regular B through Z dimensioned variable) the A-subscripted variable can still be re-dimensioned in the upward direction. A statement such as `PRINT A(36)` or `A$(40)="TEST"` is all that it takes. But, if at this point in time you make a regular DIM statement (such as `DIM B$(10)`), you set an upper limit to the A-subscript equal to its current value. This upper limit can not be revised without first invoking the `CLEAR` statement or `RUN` command.

It is interesting that the PC-1250 is a bit too easily confused as to the work space required to concatenate a string. Most PCW readers are aware that there is an 80 byte limit to the length of a concatenated string imposed by PC BASIC. But, suppose dimensioned variables `D$(0)` and `D$(1)` each have a length of 50 characters. The PC-1250 will refuse to print `LEFT$(D$(0),31)+LEFT$(D$(1),1)`. This statement brings ERROR 5 because `31+LEN(D$(1))>80`, even though only 32 bytes of work space are actually required!

Cassette Operations

A feature not mentioned in the manual is the ability of a password to be associated with the `CSAVE` command. The statement `CSAVE "TITLE","PASSW"` will save the current program in memory onto tape with the filename "TITLE". On `CLOAD`ing the program back into a PC-1250, however, the user will need to use the statement `PASS "PASSW"` in order to obtain a listing.

Another fact not mentioned in the manual is that `CSAVE`

may be used in the RSV mode to store the contents of RSV memory on tape. You may, at your option, give your RSV key assignments a file name and `CSAVE` then on tape with the PC in the RSV mode. You can then `CLOAD` these assignments back into the computer at a later date using the `CLOAD` command with the PC in the RSV mode!

Any password is ignored if it is associated with `CSAVE` in the RSV mode. If you attempt to `CLOAD` a regular program while in the RSV mode, you will obtain an ERROR 6 message. If you attempt to `CLOAD` a RSV file into program memory, you will obtain useless data.

The descriptions of the `PRINT#` and `INPUT#` statements in the manual appear to be incorrect and misleading. For example, `PRINT# "FILENAME"` does *not* save all data in the computer onto tape. It only saves fixed variables A through Z plus any A-subscripted variables. There is no single statement that will save all data onto tape.

Additionally, the statement `PRINT# "FILENAME";VARIABLE` does not save just the contents of the specified variable. It saves all A-subscripted variables starting from the named variable and continuing on up. This statement format does not work at all for regular dimensioned variables.

However, regular dimensioned variables may be saved using the form `PRINT# "FILENAME";V(*)` or `PRINT# "FILENAME";V$(*)`. The statement `INPUT# "FILENAME";X(*)` or `INPUT# "FILENAME";X$(*)` will then load the data into the appropriate X or X\$ array elements provided that X has been dimensioned appropriately and that the data types match.

Fixed, extended and dimensioned variables can be sent to tape together under one file name when using the `PRINT#` statement by separating the variable names with commas. They can later be loaded into the computer with a similar compound `INPUT#` statement. However, the pseudo-variable name `a(*)` is not permitted (as it is on the PC-1500) to store all dimensioned variables with a single specification.

The `CSAVE`, `CSAVE M`, `PRINT#` and `INPUT#` statements may all be used in a BASIC program. `CLOAD`, `CLOAD M` and `MERGE` may not be used with a program. `CHAIN` must also be invoked from within a program.

The `MERGE` command works differently in the PC-1250 from the way it works in the PC-1500. When the 1250 merges programs from tape, it does not flag the fact that a program has been merged. Thus a user "normally" has complete flexibility in running and editing merged programs. However, if line numbers are not in a natural sequence (such

as could normally be used in a single program), editing functions will decide that the unnatural sequence that occurs last in memory marks the first line of a program. RUN followed by a line number searches for the line number starting at the true start of program memory, but gives up the search when it reaches the first unnatural line sequence. RUN followed by a label, on the other hand, or a DEFINE key sequence, will not cause the search to halt when an unnatural sequence is found. In this case, the entire length of memory will be examined for the appropriate label.

When a merged program is executing, GOTO a line number or GOSUB a line number causes the interpreter to search up or down program memory (depending on whether the desired line number is greater than or less than the current line being executed). It marks the first unnatural sequence encountered as the end of search point. Using GOTO or GOSUB with a label, on the other hand, causes the entire program memory to be examined.

An analysis of this methodology indicates that Sharp displayed wisdom in their handling of merged programs in this fashion. Merged programs that have non-conflicting labels can always be executed. And, the option to go in and edit conflicting line numbers starting with the last program that was merged, enables the user to integrate programs.

Other Features

Reservable key sequences that end with the 2 symbol will execute automatically when called into the display, just as though the ENTER key had been pressed.

The 6 prefix causes the next four digits (excluding leading zeros) to be interpreted as a hexadecimal value. No decimal point is permitted in this format.

Within the range 62000<=address<=6FFFF, the statement PEEK(address) returns the decimal representation of the value residing in that RAM or ROM address. This statement has the same priority as SIN, COS, etc. Thus parenthesis are required around the address value unless it is a simple constant or variable.

POKE address,expression1,expression2,expression3,... will POKE the values corresponding to the expressions into successive RAM memory locations starting at the address location. The syntax only requires the presence of expression1 following the address.

CALL address will call a machine language routine that starts at the indicated address. Parameters can not be passed directly to/from the CPU registers as can be done on the PC-1500.

CSAVE M "TITLE";address1,address2 causes machine code to be saved on tape from address1 to address2. The title is optional. Hidden ROM (addresses 0 through 61FFF) can not be saved on tape using the CSAVE M command.

CLOAD M "TITLE";address searches for a machine language program having the indicated title on tape. When found, it loads the code into the computer starting at the indicated address. Both the title and address parameters are optional.

If a title is not specified, the next machine language program encountered on the tape will be loaded. When the address value is not specified, the program will be loaded into the addresses from which it was stored on the tape.

The PC-1250 is able to distinguish between BASIC programs, data (variables) files and machine language programs stored on tape. Thus, duplicate titles may be used for the various types of files as long as the keyword (CSAVE, PRINT# or CSAVE M) indicating the type of storage is different.

No mention is made in the manual of using the CA key after hitting the shift key. To date I have discovered the following uses for the Shift/CA sequence:

1. Clears the program cursor pointer so that CL followed by the up-arrow or down-arrow brings up the first/last line of a BASIC program when in the PRO mode.

2. Two stroke substitute for PRINT=PRINT command.

3. Two stroke substitute for TROFF command.

Note that the use of CA disables the CONT operation.

Contrary to what is stated on page 64 of the manual, the RND function does not appear to repeat a sequence of numbers each time a program is executed. In fact, it seems that the only way to repeat a RND sequence of numbers is to press the reset button. This works whether or not a key is held down to preserve program status.

Contrary again to the manual (page 81), commas and semi-colons may not be mixed within a single PRINT statement. If a comma is used it must stand alone as a separator. If not, the message ERROR 1 results.

Finally, I want to add my two cents worth to the discussion of the 1200 series' unconventional truth value assignments.

NOT must *not* be used in a conditional IF statement. NOT, OR and AND are all true Boolean operators. But, in this BASIC dialect, a statement is considered true, if and only if, it has a positive truth value. (In most BASICs, the value -1 is considered true, 0 as false.) This convention works alright for AND and OR, but not for NOT. For example:

NOT(1=2) equals NOT(00000000) which equals 11111111 (binary). In the strange yet wonderful world of two's complement arithmetic in a closed 8-bit register, this is the equivalent of -1. Therefore, IF NOT(1=2) BEEP 1 will not produce a beep when it really should.

It is better in this dialect to use IF 0=(1=2) BEEP 1.

Indeed, if 0= is substituted for NOT everywhere in a program, the desired effect can be obtained. You can use this method to translate complex logic statements from BASIC programs used on other computers.

Alternately, you could add 2 after every NOT statement. But, this method prevents the use of + in place of OR or * in place of AND (as on the old PC-1211/PC-1).

Using the 0= system saves one byte per use and gives you complete interchangeability between + and OR and * and AND. Additionally, statements will translate to any of the various PC-1500 models!

FOR HEWLETT-PACKARD HP-71B USERS

HISTOGRAM PROGRAM FOR USE WITH THINKJET PRINTER

The program presented in this article may be used to collect, organize, and analyze data for presentation as a histogram. If you do not have a Hewlett-Packard Thinkjet printer, you can use the output from the display to manually sketch a histogram chart. This is often satisfactory when "out in the field." But, if you have a Thinkjet (or similar) printer that you can connect to your HP-71B, then this program will also produce a histogram that neatly summarizes your data.

Load the program from the accompanying listing and check your work by looking for a byte count of 2981 when you CATALOG the file. When you execute the program you will get a "revolving" menu that advances to the next choice each time you elect not to choose the current option.

Basically, the program expects you to input a set of values. Statistical information is compiled from this data set (by selecting the REVIEW STATS option). This statistical information includes: the minimum, maximum, mean, standard

Listing Histogram Program for the HP-71B.

```

5 DELAY 1,.1 @ WIDTH 96 @ DESTROY ALL @
  PRINTER IS * @ PWIDTH 80
10 G=1 @ DIM X(255),F(45),S$(80)
20 DISP "HISTOGRAM"
40 DELAY 1,.1 @ FIX 2 @ DISP "HISTOGRAM
  MENU" @ AS="" @ BS=""
45 B=1 @ INPUT "ENTER DATA? ",N';B$ @ I
  F UPRC$(B$(1,1))="Y" THEN 55
46 B=2 @ INPUT "REVIEW STATS? ",N';B$ @
  IF UPRC$(B$(1,1))="Y" THEN 55
47 B=3 @ INPUT "CALCULATE CELLS? ",N';B$
  @ IF UPRC$(B$(1,1))="Y" THEN 55
48 B=4 @ INPUT "PRINT HISTOGRAM? ",N';B$
  @ IF UPRC$(B$(1,1))="Y" THEN 55
49 B=5 @ INPUT "REVIEW DATA? ",N';B$ @
  IF UPRC$(B$(1,1))="Y" THEN 55
50 B=6 @ INPUT "HELP? ",N';B$ @ IF UPRC
  $(B$(1,1))="Y" THEN 55
51 GOTO 45
55 ON B GOTO 60,90,220,410,710,780
60 DELAY 2,.1 @ PRINT "ENTER DATA  Q=Qu
  it"
70 FIX 0 @ G$=STR$(G) @ G$=G$(1,LEN(G$)-
  1) @ PRINT "DATA ("&G$&")? "; @ INPUT ""
  ,Q';Z$
75 IF Z$="Q" THEN 40 ELSE X(G)=VAL(Z$)
80 G=G+1 @ GOTO 70
90 GOSUB 930 @ DELAY .05,.05 @ PRINT @ P
  RINT
95 DELAY 2,.1 @ PRINT "STATISTICS" @ DIS
  P "CALCULATING STATS" @ DELAY 8,.5
100 FOR J=1 TO 6 @ FOR Q=1 TO 6-J
110 IF X(Q)>X(Q+1) THEN T=X(Q) @ X(Q)=X(
  Q+1) @ X(Q+1)=T
120 NEXT Q @ NEXT J @ J=J-1
130 Z=0 @ S=0 @ FOR I=2 TO J @ S=S+X(I)
  @ NEXT I @ M=S/(J-1)
140 FOR I=2 TO J @ Z=Z+X(I)*X(I) @ NEXT
  I
150 V=(Z-(J-1)*M*M)/(J-2)
160 PRINT "DATA MIN=";X(2)
170 PRINT "DATA MAX=";X(J)
180 PRINT "MEAN =" ;M @ PRINT "STD DEV
  =" ;SQR(V)
190 PRINT "VARIANCE=" ;V
200 PRINT "ENTRIES =" ;G-1
210 PRINTER IS * @ GOTO 40
220 INPUT "CELL WIDTH? ",I';W @ W=ABS(
  NT(W)) @ DELAY 2,.1 @ DISP "CALCULATING
  CELLS"
225 IF W=0 THEN BEEP @ DISP "WIDTH=0 IS
  RIDICULOUS" @ GOTO 220
230 Z=INT(X(J)+1)-INT(X(2)) @ Y=Z/W @ IF
  MOD(Z,W)>0 THEN Y=Y+1
235 IF Y>28 THEN BEEP @ DISP "TOO MANY C
  ELLS!" @ GOTO 220
240 FOR I=0 TO Y-1 @ F(I)=0 @ NEXT I
250 FOR I=0 TO Y-1
260 FOR Q=2 TO J
270 R=INT(X(2))
280 IF X(Q)>R+I*W AND X(Q)<R+(I+1)*W TH
  EN F(I)=F(I)+1
310 NEXT Q @ NEXT I
320 A=0 @ M=0
330 FOR I=0 TO Y
340 IF A<F(I) THEN A=F(I)
350 M=M+F(I)
360 NEXT I
370 D=A
380 GOTO 40
410 DISP "WATCH THE PRINTER!" @ PRINTER
  IS :PRINTER
420 PRINT CHR$(12) @ PRINT
430 PRINT " - * * * - - - H
  I S T O G R A M - * * * - - "
440 PRINT
450 U1=0 @ GOSUB 640 @ N=0 @ N2=(S6-Y)/Y
460 FOR I=Y TO 1 STEP -1
465 S$=STR$(INT(X(J)+1)-N*W) @ N=N+1 @ F
  OR L=LEN(S$)+1 TO 11 @ S$=S$&" " @ NEXT
  L

```

```

470 S$=S$&"I" @ PRINT S$; @ S$=""
475 IF I>0 THEN U=INT(60*F(I-1)/D)
480 IF U<U1 THEN U2=U1 ELSE U2=U
485 U1=U
490 GOSUB 620
500 FOR N1=1 TO N2 @ PRINT "
  "; @ GOSUB 650 @ NEXT N1
510 NEXT I
520 S$=STR$(INT(X(2))) @ FOR L=LEN(S$)+1
  TO 11 @ S$=S$&" " @ NEXT L
525 S$=S$&"I" @ PRINT S$; @ S$="" @ U2=U
  1 @ GOSUB 620
530 U1=0 @ GOSUB 640
540 FIX 0 @ PRINT "
  0%";
550 FOR I=2 TO 10 STEP 2 @ G$=STR$(INT(D
  /M*I*10+.5)) @ G$=G$(1,LEN(G$)-1)
560 IF LEN(G$)<2 THEN G$="0"&G$
570 G$=G$&"%" @ PRINT "
  ";G$;
580 NEXT I @ PRINT
600 PRINT @ GOSUB 700 @ PRINT "MAX FREQ.
  =" ;D; " CELL WIDTH=" ;W
610 PRINT CHR$(12) @ PRINTER IS * @ GOTO
  40
620 FOR K=1 TO U2 @ S$=S$&"-" @ NEXT K @
  PRINT S$ @ S$=""
630 RETURN
640 PRINT "
  I";
650 J1=1
660 FOR I1=2 TO 10 STEP 2
665 IF J1=U1 THEN PRINT "I"; @ GOTO 680
670 IF J1<I1 THEN PRINT " "; @ GOTO 68
  0
675 IF J1=6*I1 THEN PRINT ":";
680 J1=J1+1 @ IF J1>6*I1 THEN 690 ELSE 6
  55

```

```

690 NEXT I1 @ PRINT @ RETURN
700 FOR I=1 TO 20 @ PRINT " "; @ NEXT I
  @ RETURN
710 GOSUB 930 @ DELAY .05,.05 @ PRINT @
  PRINT
715 DELAY 2,.1 @ PRINT "
  CELL DATA" @
  DELAY .05,.05 @ PRINT
720 DELAY 8,.5 @ FOR I=0 TO Y-1 @ FIX 0
  @ G$=STR$(I+1) @ G$=G$(1,LEN(G$)-1)
725 PRINT "CELL, ";G$; " =";F(I);R+I*W @
  NEXT I
730 DELAY .05,.05 @ PRINT @ PRINT @ DELA
  Y 8,.5
740 PRINT "RAW DATA LISTING" @ DELAY .05
  ,.05 @ PRINT @ DELAY 8,.5
750 FOR I=2 TO 6 @ FIX 0 @ G$=STR$(I-1)
  @ G$=G$(1,LEN(G$)-1)
760 PRINT "DATA PT. # ";G$; @ FIX 2 @ PR
  INT " =" ;X(I) @ NEXT I
770 GOTO 40
780 DELAY 8,.5 @ DISP "ENTER UP TO 255 D
  ATA" @ DISP "POINTS. THEN SELECT"
790 DISP "'REVIEW STATS' BEFORE" @ DISP
  "YOU ASK TO 'PRINT'"
800 DISP "HISTOGRAM". USE 'REVIEW' @ DIS
  P "DATA' OPTION LAST."
810 DISP "YOU CAN ADD MORE DATA" @ DISP
  "BY SELECTING THE"
820 DISP "'ENTER DATA' OPTION" @ DISP "W
  HENEVER DESIRED."
830 GOTO 40
930 INPUT "USE PRINTER (Y/N)? ",N';Z$
940 IF UPRC$(Z$(1,1))="N" THEN PRINTER I
  S * ELSE PRINTER IS :PRINTER
950 RETURN

```

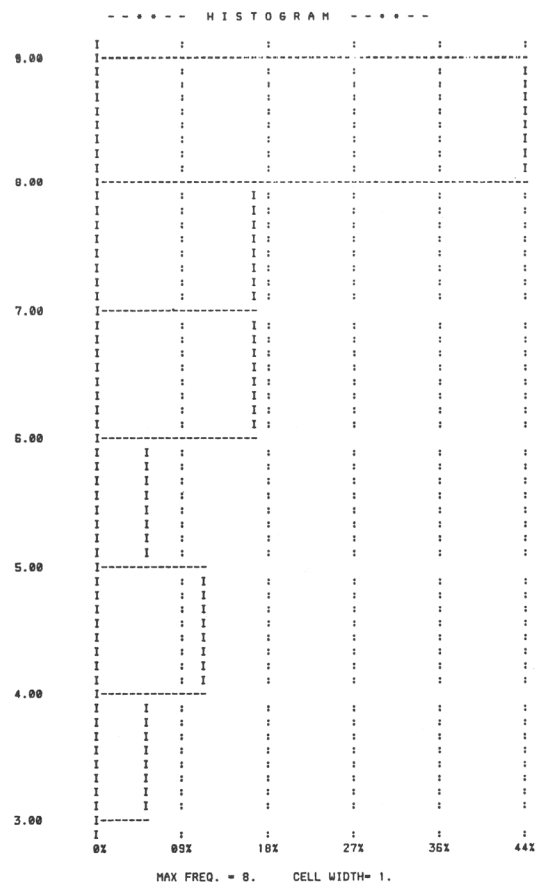
STATISTICS
 DATA MIN= 3.90
 DATA MAX= 8.50
 MEAN = 6.96
 STD DEV = 1.51
 VARIANCE= 2.28
 ENTRIES = 18.00

CELL DATA

CELL 1 = 1. 3.
 CELL 2 = 2. 4.
 CELL 3 = 1. 5.
 CELL 4 = 3. 6.
 CELL 5 = 3. 7.
 CELL 6 = 8. 8.

RAW DATA LISTING

DATA PT. # 1 = 3.90
 DATA PT. # 2 = 4.20
 DATA PT. # 3 = 4.40
 DATA PT. # 4 = 5.70
 DATA PT. # 5 = 6.30
 DATA PT. # 6 = 6.40
 DATA PT. # 7 = 6.60
 DATA PT. # 8 = 7.10
 DATA PT. # 9 = 7.70
 DATA PT. # 10 = 7.90
 DATA PT. # 11 = 8.00
 DATA PT. # 12 = 8.00
 DATA PT. # 13 = 8.10
 DATA PT. # 14 = 8.10
 DATA PT. # 15 = 8.10
 DATA PT. # 16 = 8.10
 DATA PT. # 17 = 8.20
 DATA PT. # 18 = 8.50



deviation, variance, and number of data points. The data can then be organized by sorting the data into equally spaced zones called cells (by selecting the CALCULATE CELLS option). You define the "width" of these cells or bands. The computer counts the number of data points that fall into each cell. You can have this information diagrammed as a histogram by selecting the PRINT HISTOGRAM option, provided you have a Thinkjet or similar printer connected via an HP-IL interface. If you do not have a printer, use the REVIEW DATA option to have the cell information shown on the display. There is even a HELP option that provides a summary of how to proceed, for first-time users of the program.

Note that you should proceed in the fashion: enter data,

review statistics, and calculate cells before attempting to obtain a histogram printout or review the histogram data.

A nice feature of the program is that you can put in additional data points at any time by re-selecting the ENTER DATA option! You can have a maximum of 255 data points (provided you have enough RAM available in your system). Be sure, however, to redo the statistics and cell calculations, if you do add in new data points!

The accompanying program listing includes a small sample data base you can use for checking purposes and a reduced illustration of how the histogram will appear.

This program was adapted to the HP-71B by drawing upon a version for the Sharp PC-1500 crafted by *Russ Dougherty*.

FOR PC-1500 & PC-2 USERS

VERSATILE ROOT FINDER PROGRAM

Rick Wenger, 6221 -18th Avenue, Kenosha, WI 53140, usually concentrates on the PC-1250/1350 models. However, the program he provides here is so useful and general-purpose, that he has provided versions for the PC-1500/PC-1500A as well as the PC-1250/60/1350 series.

This short program emulates the "SOLVE" function on some Hewlett-Packard top of the line programmable calculators. It is so useful that many mathematicians, scientists, engineers and students may will likely find it worthwhile to keep it handy in memory all the time.

To use the program, you define F as a function of X using any of the line numbers from 100 through 198. This function *must* assign a unique value to F for any X within the valid range of the function.

Here are a few examples of such functions:

1. $F = (X^2 - 2) * X - 5$
2. $F = (((8 * X - 75) * X + 275) * X - 485) * X + 387) * X - 90$
3. $F = X * \sin(1/X)$
4. $F = ((2 * X - 11) * X + 16) * X - 7$
5. $F = \text{SQR}(\text{ABS}(X))$

Functions 1, 2 and 4 above are polynomials that have been written in nested form to facilitate speed and accuracy during computation. Function 1 is included in the list for historical reasons. It was the original equation used to illustrate Newton's method of approximating real roots. It has one real root at 2.094551482. Equation 2 is a fifth degree polynomial with three real roots at 0.375, 2 and 3. Fifth degree polynomials can not, in general, be solved by algebraic means. Some sort of iterative process is required.

Equation 3 is a tough function for most root-finding programs. The algorithm used in this program, however, will handle it nicely, provided the user specifies some appropriate initial guesses. The function is continuous everywhere except at $X=0$. It has an infinite number of solutions in any interval which contains 0. When in the degree mode, $1/(180 * K)$ is a solution for $K=1,2,3...$ to infinity!

Equation 4 will be used to demonstrate some unique features of the algorithm used in this program. To try it out, enter the equation as a statement within the program having a line number between 100 and 198 (it is included as line 100 in the accompanying program listing). Switch your PC to the RUN mode and use the program to evaluate the function for several values of X. Your results should generally agree with the accompanying illustrative graph.

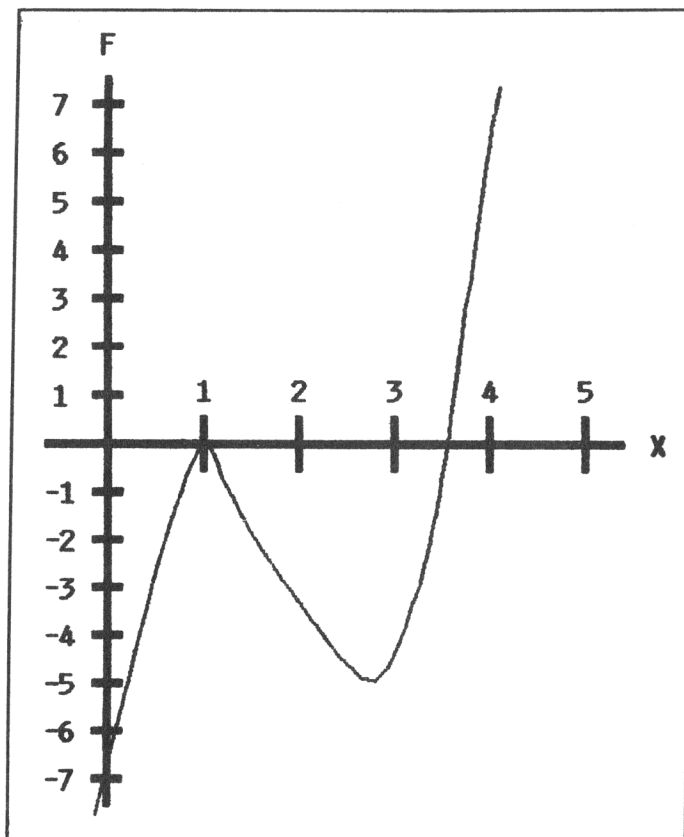
This root finder program combines the certainty of the bisection method with the ultimate speed of the secant method (a.k.a. Regula Falsi) for smooth functions. The program does not require that the two initial guesses bracket the root. (Indeed, this is not always recommended.) However, if a root can be bracketed (with $F(x_1)$ opposite in sign to $F(x_2)$), the program will always converge in the

region of the desired root. Accuracy is typically the best that is attainable using floating point firmware.

For example, when evaluating equation 4, the starting guesses $X=5$ and $X=1.5$ will cause the root finder to converge on the root $X=3.5$ after 9 iterations. The bisection method would have required 32. The secant method takes 24 iterations to arrive at the "wrong" root at $X=1$.

It is possible to use this program to find the root at $X=1$ as well, if desired. However, since the X-axis is tangential to the curve at this juncture, it is better not to bracket the root. Instead, choose two guesses to one side of the root such as 0.5 and 0.75. Otherwise the program may converge on the already calculated root of $X=3.5$. It will be observed

Graph $F(x) = 2X^3 - 11X^2 + 16X - 7$.



that a loss of accuracy occurs because of the flatness of the curve in the vicinity of $X=1$. Any X in the range plus/minus $1E-5$ will yield $F(x)=0$.

It should also be noted that there is no such thing as a foolproof root-solving program. As a direct consequence of the flexibility and speed afforded by the algorithm used in this program, false roots can occur in certain rare instances. As an example of this, try the guesses 0.51835 and 1.5 on the equation labeled number 2 in the list of example functions.

Equation number 5 is given as an example of a function that the program does not help solve. Notice that the function lies completely on one side of the X -axis and that

Program Root-Finder for the PC-1500.

```

10: CLEAR
20: INPUT "x? "; Z:
   GOSUB 99: PRINT
   "f(x)= "; F:
   GOTO 20
30: IF (ABS (F-E)<
   =.1*ABS F) OR (
   C<2) LET Z=W+W-
   X+C: GOSUB 99:
   GOTO 30
40: IF SGN F<>SGN
   E LET V=W:D=E
50: Z=(E*X-F*W)/(E
   -F): IF Z=X
   PRINT "x= "; X:
   PRINT "f(x)= "
   ; F: GOTO 10
60: IF (D=0) OR (
   SGN (X-Z)<>SGN
   (V-Z)) GOSUB 99
   : IF F<>E GOTO 4
   0
70: IF D=0 GOTO 30
80: Z=V/2+X/2:
   GOSUB 99: IF
   SGN F=SGN E LET
   W=V:E=D
90: GOTO 40
99: W=X:E=F:X=Z:C=
   C+1
100: F=2*X^3-11*X^2
   +16*X-7
199: RETURN

```

$F'(x)$ is undefined at $X=0$, which is where the root resides!

A few features worth noting have been incorporated into the program to provide convenience in its operation. You may elect not to make any guesses, in which case the default inputs will be $X=0$ and $X=1$. Alternately, you may make just a single guess, or set the last 2 guesses equal to one another without disturbing the program.

The variable C has been set aside as a function evaluation counter. This, incidentally, is the proper criterion for judging the efficiency of any equation solver. Newton's method, for example, often requires few iterations to converge, but is not a good choice for a general root-finder. This is because three function evaluations are required per iteration in order to obtain each new estimate of the root. Newton's method also has problems with convergence in many instances.

There are two program listings shown. PC-1250/60 users should eliminate the WAIT statements in lines 10 and 50 of the PC-1250/60/1350 version. Users who want to watch a function as it converges can add a line such as: 198 PAUSE X, F to the program. Enjoy!

Program Root-Finder for the PC-1250/60/1350.

```

10: CLEAR : WAIT 0
20: INPUT "x? "; Z:
   GOSUB 99: PRINT "f(x
   )= "; F: GOTO 20
30: IF ABS (F-E)<=.1*
   ABS F OR C<2 LET Z=W
   +W-X+C: GOSUB 99:
   GOTO 30
40: IF SGN F<>SGN E
   LET V=W:D=E
50: Z=(E*X-F*W)/(E-F):
   IF Z=X PRINT "x= "; X
   : WAIT : PRINT "f(x)
   = "; F: GOTO 10
60: IF D=0 OR SGN (X-Z)<
   >SGN (V-Z) GOSUB 99
   : IF F<>E GOTO 40
70: IF D=0 GOTO 30
80: Z=V/2+X/2: GOSUB 99:
   IF SGN F=SGN E LET
   W=V:E=D
90: GOTO 40
99: W=X:E=F:X=Z:C=C+1
100: F=2*X^3-11*X^2+16*X-
   7
199: RETURN

```

FOR PC-1350 USERS

PC-1350 MONITOR

Hey, look what happens with teamwork! *Norlin Rober* adapted *Rick Wenger's* PC-1250 Monitor so that it would run on the PC-1350. At the time this originally occurred, Rick did not

have a PC-1350 of his own. Now he does. This article he submitted goes a long ways towards introducing newcomers to the use of a monitor program. Rick's address is 6221 - 18th Avenue, Kenosha, WI 53140.

About This Machine Language Monitor

This program was originally written for a PC-1250 having only 2K of RAM. It has been adapted to the PC-1350 by Norlin Rober. He requested that I write this article that describes its use and application.

Since the original program had to fit in such a small space, there are no elaborate prompts or esoteric, seldom-used functions. It operates in an essentially modelless environment. Most of the user defined keys have dual functions through the convention of preceding defined key sequences by BRK and an argument to the function. It is a dandy little program, however, especially for PC-1350 owners who do not have any RAM card expansion memory. On the other hand, those with the interest should find it easily upgradeable to include capabilities such as word search, block moving, etc.

Note the simple method used to protect the machine language portion of the program. We simply move up the value in the "start of BASIC" pointer and execute NEW. This is good news! The PC-1350 is not as hostile to the machine language programmer as the PC-1250 seems to be.

Loading and Saving the Monitor

To key in the program from the accompanying listings:

1. POKE 66F01,6D0,665
 2. Execute NEW
 3. Key in the machine language portion using POKE for the address range 664DA through 664F9.
 4. Key in the BASIC part of the program. When through keying in the BASIC portion, MEM should yield 1107.
 5. The rest of the machine language for the monitor can be entered using the DEF M capability of the monitor itself.
- Once the program has been entered, you can save a copy of it on tape:

1. CSAVE M 6640F,665CE
2. CSAVE

To reload the program from tape in the future:

1. POKE 66F01,6D0,665
2. Execute NEW
3. CLOAD M
4. CLOAD

To restore the PC-1350 to its normal memory configuration when through using the monitor: CALL 0.

Here is how the PC-1350 Monitor utilizes memory:

66031-640E: Area for user machine language routines, protected from BASIC (990 bytes).

6640F-65CE: Machine language portion of the Monitor. Includes the area 664FC-FE used for temporary CPU registers and 66500-5F used as a temporary CPA.

66500-67DC: BASIC area used by BASIC portion of the Monitor program.

667DD-6C2F: Remaining BASIC area (1107 bytes) available for BASIC routines to the user.

Fixed variables A, B, C and D are used by this Monitor program and should not be disturbed by user routines.

Monitor Instructions

Monitor operations are invoked by use of DEFined key sequences. That is, pressing of the DEF key followed by one of the character keys shown in the listing below. Most of these operations are preceded by entering a value into the display.

DEF M: Preceded by a 4-digit hexadecimal address (note that this address value is *not* prefixed with the & sign) puts the program into the monitor mode. Thereafter you may move to any address by simply keying a 4-digit address and hitting the ENTER key. Enter without any input increments the monitor address by four. You can change this increment within the range 0 through 9 by keying a single digit followed by ENTER. An increment of 5 is useful when viewing the contents of memory. An increment of 1 is useful when entering machine code into memory. Entering machine code

into RAM is accomplished by merely positioning the monitor at the desired starting address, then keying in each 2-digit hexadecimal value (without the & prefix) followed by ENTER.

DEF SPC: Decrements the monitor address by the current increment value.

DEF F: Preceded by BRK and 2 digits, this fills the temporary CPA with the specified hexadecimal value. When not preceded by BRK and 2 digits, it simply moves the monitor to the temporary registers area.

DEF B: Puts a breakpoint at the current monitor address.

DEF D: Deletes just the last breakpoint encountered.

DEF J: Preceded by BRK and 4 digits, causes a jump to the specified hexadecimal address. When not prefixed by BRK and an address value, it causes a jump to the previously entered jump address unless used immediately after DEF D. In the latter case it serves as a continue function. You *must* use DEF J if a breakpoint is present a program that will be executed by the monitor. You *must not* use DEF J if the routine ends with an RTS instruction!

DEF S: Preceded by BRK and 2 digits, it searches for the specified hexadecimal byte starting from the current monitor location. If not preceded by BRK and 2 digits, it resumes the search from the current location for the last specified data byte.

DEF H: Provides a hardcopy of memory from the current monitor location to an external printer. Output begins at the current monitor location. Unless this option is preceded by BRK and 4 digits representing an ending address, printing will continue until the BRK key is pressed or the top of memory (address 6FFFF) is reached.

DEF =: Converts the decimal number currently in the display into a 4-digit hexadecimal equivalent.

CALL 66473: This is a pseudo-NEW function. It effectively "news" the BASIC programming area while retaining the Monitor program. It can also be used immediately after an inadvertent regular NEW or a bad "crash" to resurrect the Monitor program from the graveyard!

Putting Monitor Through Its Paces

First, let's become familiar with moving around in memory using the Monitor. Key 6500 DEF M (note that the term DEF stands for using the DEFine key, not the letters D..E..F!) Remember you do not key in the & prefix when entering values to the monitor program. What you will now see appear on your display is the first four bytes of the BASIC language portion of the Monitor program, in the left-hand portion of the display. Preceding that is the address of the first byte in this 4-byte group. To the far left is the antecedent byte, from address 65CF. You will find, when entering machine code, that it is often helpful to be able to see what you have just done.

You may wish at this point to explore the computer's method of storing BASIC programs. The increment value (for the number of bytes advanced/displayed) is initially set to four. You can revise this within the range 0 to 9 by keying a single digit before hitting the ENTER key. Do *not* enter two digits, however, or you will start modifying the BASIC portion of the Monitor program itself. You can move backwards in memory by using the DEF SPC option.

Another interesting section of memory to explore is within the address range 0000 through 1FFF. This is the area of the so-called "hidden" ROM.

Note that you can move to any new address when in the monitor mode by just keying in a 4-digit address and then pressing the ENTER key.

Try A Simple Program

Since most microprocessors do not have multiplication capabilities in their instruction set, a good first program to develop is a subroutine that accomplishes this task. We will attempt to write such a program using the Monitor to illustrate both code entry and debugging operations. The method demonstrated may be referred to by some as the "brute

Program BASIC Portion of the PC-1350 Monitor.

```

1:"Y" CALL &65B3:D=?
?: RETURN
2:GOSUB "Y": POKE C,D:
C=C+1
3:"B= INT (C/256):
POKE &64FA,C-256*B,B
: CALL &6560
4:B=0: INPUT "12 1234:
12 12 12 12 ";B$:
ON LEN B$ GOTO 11,2,
3,6
5:C=C+A: GOTO "
6:"M" AREAD B$: GOSUB
"Y":C=D: POKE &662A,
&DF:A=4: GOTO "
7:"J" AREAD B$: IF
LEN B$=4 GOSUB "Y":B
= INT (D/256): POKE
&64B5,B,D-256*B
8:CALL &6499: GOTO 10
9:"F" AREAD B$: IF
LEN B$=2 GOSUB "Y":B
= INT (D/256): POKE
&646B,D: CALL &646A
10:C=&64FC: GOTO "
11:A= VAL B$: GOTO 5
12:" "C=C-A: GOTO "
13:"B" CALL &6447:
GOTO "
14:"D" CALL &6432:
GOTO "
15:"=" AREAD D:B= INT (
D/256): POKE &64FA,D
-256*B,B: CALL &64DA
: PRINT B$: GOTO "
16:"S" AREAD B$: IF
LEN B$=2 GOSUB "Y":
POKE &6426,D
17:CALL &640F:C= PEEK &
64FA+256* PEEK &64FB
: GOTO "
18:"H" AREAD B$: IF
LEN B$=4 GOSUB "Y":C
=D
19:A=5: POKE &662A,&E2:
GOTO "

```

force" technique. The operands will be restricted to the range 00 - FF with the product thus limited to the range 0000 through FFFF hexadecimal.

The first step we will perform (following a top-down structure) is to write a BASIC routine to handle I/O for the user and for calling the machine language instructions. The machine language code will perform the actual multiplication. Here is the BASIC code for our little adventure:

```

100 "MULTIPLY"INPUT "MULTIPLICAND (0-255): ";N:POKE
66031,N
110 INPUT "MULTIPLIER (0-255): ";M:POKE 66032,M

```

Program Machine Language Parts of the PC-1350 Monitor.

640F:10 64 FA 84	64BF:52 21 11 FD
6413:1A 00 00 04	64C3:52 22 11 FE
6417:88 55 85 63	64C7:52 80 10 65
641B:40 2A 07 82	64CB:00 53 11 02
641F:13 04 0A 88	64CF:57 11 01 00
6423:35 88 63 00	64D3:5E 81 19 02
6427:29 12 10 64	64D7:58 32 37 84
642B:FA 84 1B 37	64DB:10 64 FA 1A
642F:37 37 37 10	64DF:84 78 65 A1
6433:64 2F 82 00	64E3:82 10 6C F3
6437:02 18 10 00	64E7:1B 85 78 65
643B:00 82 19 10	64EB:A1 82 11 F1
643F:64 3A 1A 85	64EF:1B 02 F5 11
6443:11 B5 1B 37	64F3:F0 52 23 11
6447:10 64 FA 86	64F7:F5 52 37
644B:1A 84 02 B7	
644F:DB 02 64 DA	6560:78 64 DA 88
6453:02 79 82 00	6564:00 03 10 6C
6457:02 07 06 19	6568:F1 18 88 10
645B:10 64 3B 86	656C:66 2F 19 85
645F:53 11 3A 87	6570:00 04 63 E0
6463:53 10 64 2F	6574:05 2A 09 82
6467:82 19 37 02	6578:13 04 0A 88
646B:00 00 60 10	657C:35 2C 03 88
646F:64 FF 1F 37	6580:18 88 78 65
6473:02 DC 03 67	6584:A1 10 66 2C
6477:82 10 6F 03	6588:82 1B 02 66
647B:1B 10 65 D1	658C:87 DB 02 33
647F:D4 00 11 D0	6590:86 DB 89 00
6483:D5 FF 10 67	6594:03 78 65 A1
6487:DC D5 FF 10	6598:26 DA 26 06
648B:6F 01 02 D0	659C:50 41 2B 0A
648F:03 65 82 1B	65A0:37 59 78 65
6493:37 37 37 37	65A4:A8 DA 59 58
6497:37 37 80 00	65A8:64 0F 74 30
649B:5F 10 65 00	65AC:67 3A 3A 03
649F:19 11 02 52	65B0:74 07 37 00
64A3:10 64 FC 57	65B4:04 84 02 30
64A7:30 11 FD 57	65B8:1E 88 10 6C
64AB:31 11 FE 57	65BC:F1 18 87 50
64AF:32 10 65 02	65C0:63 00 29 04
64B3:57 79 00 00	65C4:51 51 51 51
64B7:10 65 02 52	65C8:10 65 E1 00
64BB:20 10 64 FC	65CC:03 19 37

```

120 CALL 66100
130 PRINT "PRODUCT= ";PEEK 66033+256*PEEK 66034
140 GOTO 100

```

The structure of the BASIC routine shows that certain details of the upcoming machine language routine have already been determined. Thus, the starting address of the machine code must be at 66100. We have designated locations 66031 and 66032 as being the input (multiplicand and multiplier) values. Locations 66033 + 66034 will hold the result of the multiplication process as a 2-register, 16-bit value. It is important to remember when working with this CPU that it is generally more convenient to have the low order byte of a 16-bit value precede the high byte when stored in memory.

Using this information, here is a machine language routine that might seem to do the job:

Addr	Code	Labels	Mnenonics	Comments
6100	90	START	SLP 10	Zero out 10
6101	60 00		AND (P) 00	and 11 to
6103	50		INC P	hold product.
6104	60 00		AND (P) 00	
6106	10 60 31		LDW 6031	
6109	57		LDA (W)	Now load AB
610A	03 00		LDB 0	with multiplicand.
610C	88		SLP 08	Load (08)
610D	11 32		LDWL 32	with
610F	55		LD (P)(W)	multiplier.
6110	90		SLP 10	Next, add
6111	49	LOOP	DE (08)	AB to
6112	3A 04		FCS BYE	(10)-(11)
6114	14		ADD (P+) AB	a total
6115	20 05		REV LOOP	of (08) times.
6117	10 60 33 BYE		LDW 6033	Offload
611A	90		SLP 10	product in (10)-(11)
611B	18		EX1 (P)(W)	to 6033-34.
611C	37		RTS	Return to BASIC.

To enter the machine code into memory, simply position the monitor to address 6100 and key in each 2-digit hexadecimal code followed by ENTER.

To execute the program, key BRK to exit the monitor. Then issues the command RUN "MULTIPLY". Respond to the prompts to key: 2 ENTER 2 ENTER. (To solve for 2 times 2....) Oops! Did you get a response of 514? Obviously, something is wrong.

Since this is a tutorial discussion, the above program is deliberately incorrect for illustrative reasons. Now to explore further. What could be wrong?

The first thing that might come to mind is that maybe we goofed on an address in the BASIC program. Let's explore that possibility. Key: 6031 DEF M. What do you see there? How about 02 02 02 02? Well, 0202 is indeed the hexadecimal equivalent of 514 decimal. So, it doesn't appear that the problem lies with the BASIC portion of our program.

Onward we snoop. Lets try running the program again for more clues. Key: BRK, RUN "MULTIPLY" and press ENTER. Respond to the prompts with 3 ENTER, 4 ENTER. Check locations 6031 to 6034 using the monitor. What happened now? Does it look interesting?

Since we now know that the problem lies within the machine language routine, we can try setting up a breakpoint to attempt to pinpoint the location and reason for the difficulty. The code from address 6100 to 610F is what might be called "set up" business. It is straightforward in nature. We are getting the stack ready for the real business of multiplication, which gets done by the instructions at addresses 6110 to 6116. Let's enter a breakpoint at address 6110 to see if the "set up" portion is performing as expected.

We set the breakpoint by positioning the monitor to 6110 then keying DEF B. Locations 6110 to 6112 with then be overwritten by a breakpoint instruction. This instruction directs the computer to jump to a monitor routine that will in turn allow us to examine the CPU and CPA registers.

Now key: BRK 6100 DEF J. (Remember, we *must* use DEF J when a breakpoint is in place.) Since 6031 and 6032 still contain 3 and 4 from our previous running of the program, these will remain the input values to the machine language routine. Once you have keyed the BRK 6100 DEF J sequence, the monitor should come back almost immediately, positioned at 64FC. The range 64FC through 64FE is the temporary CPU area. The locations are assigned as follows:

64FC = CPU Register P
64FD = CPU Register P'
64FE = CPU Register SP

POCKET COMPUTER NEWSLETTER

P.O. Box 232, Seymour, CT 06483

In addition, the entire area from 6500 to 655F is dedicated to representing the CPA at breakpoint. Here is a technical explanation of how the temporary CPU area and the temporary CPA area are utilized with DEF J when a breakpoint is in place.

When DEF J is invoked, the temporary CPU register locations 64FC - FE are dumped to the real CPU registers and the temporary CPA is dumped to the real CPA. Control then passes to the specified jump address where instructions are executed until a breakpoint is reached. At this point the temporary CPA and CPU registers are made to reflect the actual CPU and CPA contents. The actual CPA and CPU registers are then stuffed with "safe" values so that the BASIC operating system will not crash. Control then passes to the monitor which points itself to the temporary CPU area.

It seems to take quite a few words to explain what is a relatively simple concept. Really, all we need to worry about now is that the code in our routine between address 6100 and 610F had the desired effect on the CPA registers involved in the operations. Now, let's make a list of what we should be expecting in the CPA registers when we reach the breakpoint we have set.

CPA Register A is at 6502, expected value: 03
CPA Register B is at 6503, expected value: 00
CPA register (08) is at 6508, expected value: 04
CPA register (10) is at 6510, expected value: 00
CPA register (11) is at 6511, expected value: 00

Now, you check those memory locations to see if they contain the expected values. Well, if you are doing the same things we have explained, then they do! This indicates that the first part of our routine seems to be working OK.

Let's delete the breakpoint at 6110. Key DEF D to remove it. Now position the monitor to 6117 and key DEF B to set a new breakpoint at that location. Key DEF J to execute the 6110 to 6117 portion of the machine language routine. (Note that this Monitor program is smart enough to know that since we recently deleted a breakpoint and have not specified a new jump address, it is to continue operations from the old breakpoint location!)

Once you have boldly executed the DEF J option without entering an argument to it, you can check the temporary registers area to see if they are as expected. The first thing you will see is: 64FC = 14. What CPU register does this represent? Is this what you expected it to be? Next check out the A, B, (08), (10) and (11) CPA locations as before.

At this point, a light may begin to dawn. If not, review the explanation under the ADD (P+) AB instruction on page 4 of PCW Issue 38. What have we neglected to do?

We have neglected to reposition P to (10) for each pass through the loop! The need for constantly updating the P pointer is a quirk of this particular CPU and a common source of frustration.

The fix is simple. Use the monitor to change location 6116 so that it contains 06 (instead of 05). Then, on each pass through the loop, the program will reverse branch back one more instruction and position P properly.

Before you try out the corrected program, be sure to remove the remaining breakpoint at 6117 by keying DEF D.

Now you can multiply!

If you have stayed this far, then you will find the rest of the monitor options easy. Here are a few more tips to ease your use of the program.

The DEF = option provides decimal to hexadecimal conversion. Key 4321 DEF = and observe 10E1. To see what nn value would be required for a reverse relative branch from 6116 to 6110, key 6116-6110 DEF =.

DEF F is a convenient way to fill the CPA with any desired value prior to using DEF J.

DEF S uses a machine language routine to search hundreds of times faster than would be possible using BASIC. For example, try positioning the monitor to 6100 then key the sequence: BRK 37 DEF S to find the end point of the multiplication routine.

Enjoy your new programming tool!